Contents

Customizing the installation configuration

7 minute read

Prerequisites

Before you begin, check the following prerequisites:

1. Download the Istio release.

2. Perform any necessary platform-specific setup.
3. Check the Requirements for Pods and Services.

In addition to installing any of Istio's built-in configuration profiles, `istioctl install` provides a complete API for customizing the configuration.

- The `IstioOperator` API

The configuration parameters in this API can be set individually using `--set` options on the command line. For example, to enable debug logging in a default configuration profile, use this command:

```
$ istioctl install --set values.global.logging.level=debug
```

Alternatively, the `IstioOperator` configuration can be specified in a YAML file and passed to `istioctl` using the `-f` option:

```
$ istioctl install -f samples/operator/pilot-k8s.yaml
```

> For backwards compatibility, the previous Helm installation options, with the exception of Kubernetes resource settings, are also fully supported. To set them on the command line, prepend the option name with "`values.`". For example, the following command overrides the `pilot.traceSampling` Helm configuration option:
>
> ```
> $ istioctl install --set values.pilot.traceSampling=0.1
> ```

Helm values can also be set in an `IstioOperator` CR (YAML file) as described in Customize Istio settings using the Helm API, below.

If you want to set Kubernetes resource settings, use the `IstioOperator` API as described in Customize Kubernetes settings.

Identify an Istio component

The `IstioOperator` API defines components as shown in the table below:

Components

base
pilot
ingressGateways
egressGateways
cni
istiodRemote

The configurable settings for each of these components are available in the API under `components.<component name>`. For example, to use the API to change (to false) the `enabled` setting for the `pilot` component, use `--set components.pilot.enabled=false` or set it in an `IstioOperator` resource like this:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  components:
    pilot:
      enabled: false
```

All of the components also share a common API for changing Kubernetes-specific settings, under `components.<component name>.k8s`, as described in the following section.

Customize Kubernetes settings

The `IstioOperator` API allows each component’s Kubernetes settings to be customized in a consistent way.

Each component has a

`KubernetesResourceSpec`, which allows the following settings to be changed. Use this list to identify the setting to customize:

1. Resources
2. Readiness probes
3. Replica count
4. `HorizontalPodAutoscaler`
5. `PodDisruptionBudget`
6. Pod annotations
7. Service annotations
8. `ImagePullPolicy`
9. Priority class name
10. Node selector
11. Affinity and anti-affinity
12. Service
13. Toleration
14. Strategy

5. Env
6. Pod security context

All of these Kubernetes settings use the Kubernetes API definitions, so Kubernetes documentation can be used for reference.

The following example overlay file adjusts the resources and horizontal pod autoscaling settings for Pilot:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  components:
    pilot:
      k8s:
        resources:
          requests:
            cpu: 1000m # override from default 500m
            memory: 4096Mi # ... default 2048Mi
        hpaSpec:
          maxReplicas: 10 # ... default 5
          minReplicas: 2  # ... default 1
```

Use `istioctl install` to apply the modified settings to the cluster:

```
$ istioctl install -f samples/operator/pilot-k8s.yaml
```

Customize Istio settings using the Helm API

The `IstioOperator` API includes a pass-through interface to the Helm API using the `values` field.

The following YAML file configures global and Pilot settings through the Helm API:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    pilot:
      traceSampling: 0.1 # override from 1.0
    global:
      monitoringPort: 15014
```

Some parameters will temporarily exist in both the Helm and `IstioOperator` APIs, including Kubernetes resources, namespaces and enablement settings. The Istio community recommends using the `IstioOperator` API as it is more consistent, is validated, and follows the community graduation process.

Configure gateways

Gateways are a special type of component,

since multiple ingress and egress gateways can be defined. In the `IstioOperator` API, gateways are defined as a list type. The `default` profile installs one ingress gateway, called `istio-ingressgateway`. You can inspect the default values for this gateway:

```
$ istioctl profile dump --config-path components.ingressGateways
$ istioctl profile dump --config-path values.gateways.istio-ingressgateway
```

These commands show both the `IstioOperator` and Helm settings for the gateway, which are used together to define the generated gateway resources. The built-in gateways can be customized just like any other component.

> From 1.7 onward, the gateway name must always be specified

when overlaying. Not specifying any name no longer defaults to `istio-ingressgateway` or `istio-egressgateway`.

A new user gateway can be created by adding a new list entry:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  components:
    ingressGateways:
      - name: istio-ingressgateway
        enabled: true
      - namespace: user-ingressgateway-ns
        name: ilb-gateway
        enabled: true
        k8s:
          resources:
            requests:
              cpu: 200m
          serviceAnnotations:
            cloud.google.com/load-balancer-type: "i
nternal"
          service:
            ports:
            - port: 8060
              targetPort: 8060
              name: tcp-citadel-grpc-tls
            - port: 5353
              name: tcp-dns
```

Note that Helm values (`spec.values.gateways.istio-ingressgateway/egressgateway`) are shared by

all ingress/egress gateways. If these must be customized per gateway, it is recommended to use a separate IstioOperator CR to generate a manifest for the user gateways, separate from the main Istio installation:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  profile: empty
  components:
    ingressGateways:
      - name: ilb-gateway
        namespace: user-ingressgateway-ns
        enabled: true
        # Copy settings from istio-ingressgateway a
s needed.
  values:
    gateways:
      istio-ingressgateway:
        debug: error
```

Advanced install customization

Customizing external charts and profiles

The `istioctl install`, `manifest generate` and `profile` commands can use any of the following sources for charts and profiles:

- compiled in charts. This is the default if no `--manifests` option is set. The compiled in charts are the same as those in the `manifests/` directory of the Istio release `.tgz`.
- charts in the local file system, e.g., `istioctl install --manifests istio-1.11.3/manifests`

- charts in GitHub, e.g., `istioctl install --manifests https://github.com/istio/istio/releases/download/1.11.3/istio-1.11.3-linux-arm64.tar.gz`

Local file system charts and profiles can be customized by editing the files in `manifests/`. For extensive changes, we recommend making a copy of the `manifests` directory and make changes there. Note, however, that the content layout in the `manifests` directory must be preserved.

Profiles, found under `manifests/profiles/`, can be edited and new ones added by creating new files with the desired profile name and a `.yaml` extension. `istioctl` scans the `profiles` subdirectory and all profiles found there can be referenced by name in the `IstioOperatorSpec` profile field. Built-in

profiles are overlaid on the default profile YAML before user overlays are applied. For example, you can create a new profile file called `custom1.yaml` which customizes some settings from the `default` profile, and then apply a user overlay file on top of that:

```
$ istioctl manifest generate --manifests mycharts/
--set profile=custom1 -f path-to-user-overlay.yaml
```

In this case, the `custom1.yaml` and `user-overlay.yaml` files will be overlaid on the `default.yaml` file to obtain the final values used as the input for manifest generation.

In general, creating new profiles is not necessary since a similar result can be achieved by passing multiple overlay files. For example, the command above is equivalent to passing two user overlay files:

```
$ istioctl manifest generate --manifests mycharts/
-f manifests/profiles/custom1.yaml -f path-to-user-
overlay.yaml
```

Creating a custom profile is only required if you need to refer to the profile by name through the `IstioOperatorSpec`.

Patching the output manifest

The `IstioOperator` CR, input to `istioctl`, is used to generate the output manifest containing the Kubernetes resources to be applied to the cluster. The output manifest can be further customized to add, modify or delete resources through the `IstioOperator` `overlays` API, after it is generated but before it is applied to the cluster.

The following example overlay file (`patch.yaml`) demonstrates the type of output manifest patching that can be done:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  profile: empty
  hub: docker.io/istio
  tag: 1.1.6
  components:
    pilot:
      enabled: true
      namespace: istio-control
      k8s:
        overlays:
          - kind: Deployment
            name: istiod
            patches:
              # Select list item by value
              - path: spec.template.spec.containers
.[name:discovery].args.[30m]
                value: "60m" # overridden from 30m
              # Select list item by key:value
              - path: spec.template.spec.containers
.[name:discovery].ports.[containerPort:8080].contai
nerPort
                value: 1234
              # Override with object (note | on val
```

```
ue: first line)
              - path: spec.template.spec.containers
.[name:discovery].env.[name:POD_NAMESPACE].valueFro
m
                value: |
                  fieldRef:
                    apiVersion: v2
                    fieldPath: metadata.myPath
              # Deletion of list item
              - path: spec.template.spec.containers
.[name:discovery].env.[name:REVISION]
              # Deletion of map item
              - path: spec.template.spec.containers
.[name:discovery].securityContext
          - kind: Service
            name: istiod
            patches:
              - path: spec.ports.[name:https-dns].p
ort
                value: 11111 # OVERRIDDEN
```

Passing the file to `istioctl manifest generate -f patch.yaml` applies the above patches to the default profile output manifest. The two patched resources will be modified as shown below (some parts of the resources are omitted for brevity):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: istiod
spec:
  template:
    spec:
      containers:
      - args:
        - 60m
        env:
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              apiVersion: v2
              fieldPath: metadata.myPath
        name: discovery
        ports:
        - containerPort: 1234
---
apiVersion: v1
kind: Service
metadata:
  name: istiod
spec:
  ports:
  - name: https-dns
    port: 11111
---
```

Note that the patches are applied in the given order. Each patch is applied over the output from the previous patch. Paths in patches that don’t exist in the output manifest will be created.

List item path selection

Both the `istioctl --set` flag and the `k8s.overlays` field in `IstioOperator` CR support list item selection by `[index]`, `[value]` or by `[key:value]`. The –set flag also creates any intermediate nodes in the path that are missing in the resource.